CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS
CURSO DE ENGENHARIA DE COMPUTAÇÃO
DEPARTAMENTO DE COMPUTAÇÃO

LAB. DE ALGORÍTMOS E ESTRUTURAS DE DADOS II

TRABALHO PRÁTICO 2

Turma: T02 (2023.2 - 6T34)
Alunos: Ana Clara Cunha Lopes,
Arthur Affonso Bracarense Ferreira,
Gustavo de Assis Xavier.
Data: 30/11/2023
Professor: Otaviano Martins Monteiro

BELO HORIZONTE
2023

A seguinte sequência de números é conhecida como série de Fibonacci, proposta pelo matemático italiano Leonardo Fibonacci no século XI, é tal que cada elemento (com exceção dos dois primeiros que são 0 e 1), é igual à soma dos dois anteriores: 0, 1, 1, 2, 3, 5, 8, 13, 21, 34, 55, 89, ...

```
F(1) = 0
F(2) = 1
F(n) = F(n-2) + F(n-1)        n > 2
```

1) Implemente a geração de números da Série de Fibonacci de forma **recursiva**. Execute o método até o valor que você conseguir, com o limite de 1000 (provavelmente você não vai chegar nem perto desse valor).
   a) Conte o número de chamadas recursivas para cada execução;

Resposta 1 - a)

Considerando que a implementação do código foi feita em JAVA, temos o seguinte resultado para geração de números da Série de Fibonacci de forma recursiva:

```
F(1) = 1 (Chamadas recursivas: 1)
F(2) = 1 (Chamadas recursivas: 3)
F(3) = 2 (Chamadas recursivas: 5)
F(4) = 3 (Chamadas recursivas: 9)
F(5) = 5 (Chamadas recursivas: 15)
F(6) = 8 (Chamadas recursivas: 25)
F(7) = 13 (Chamadas recursivas: 41)
F(8) = 21 (Chamadas recursivas: 67)
F(9) = 34 (Chamadas recursivas: 109)
F(10) = 55 (Chamadas recursivas: 177)
F(11) = 89 (Chamadas recursivas: 287)
F(12) = 144 (Chamadas recursivas: 465)
F(13) = 233 (Chamadas recursivas: 753)
F(14) = 377 (Chamadas recursivas: 1219)
F(15) = 610 (Chamadas recursivas: 1973)
F(16) = 987 (Chamadas recursivas: 3193)
F(17) = 1597 (Chamadas recursivas: 5167)
F(18) = 2584 (Chamadas recursivas: 8361)
F(19) = 4181 (Chamadas recursivas: 13529)
F(20) = 6765 (Chamadas recursivas: 21891)
F(21) = 10946 (Chamadas recursivas: 35421)
F(22) = 17711 (Chamadas recursivas: 57313)
F(23) = 28657 (Chamadas recursivas: 92735)
F(24) = 46368 (Chamadas recursivas: 150049)
```

```
F(25) = 75025 (Chamadas recursivas: 242785)
F(26) = 121393 (Chamadas recursivas: 392835)
F(27) = 196418 (Chamadas recursivas: 635621)
F(28) = 317811 (Chamadas recursivas: 1028457)
F(29) = 514229 (Chamadas recursivas: 1664079)
F(30) = 832040 (Chamadas recursivas: 2692537)
F(31) = 1346269 (Chamadas recursivas: 4356617)
F(32) = 2178309 (Chamadas recursivas: 7049155)
F(33) = 3524578 (Chamadas recursivas: 11405773)
F(34) = 5702887 (Chamadas recursivas: 18454929)
F(35) = 9227465 (Chamadas recursivas: 29860703)
F(36) = 14930352 (Chamadas recursivas: 48315633)
F(37) = 24157817 (Chamadas recursivas: 78176337)
F(38) = 39088169 (Chamadas recursivas: 126491971)
F(39) = 63245986 (Chamadas recursivas: 204668309)
F(40) = 102334155 (Chamadas recursivas: 331160281)
```

Para o resultado acima, foi considerando que n = 40, pois a partir desse valor, o código fica com a execução muito lenta. A abordagem recursiva tradicional da série de Fibonacci é ineficiente para valores grandes de n.

---

b) Gere um gráfico de **`n x número de chamadas recursivas`**. Verifique que é uma função exponencial;

Resposta 1 - b)

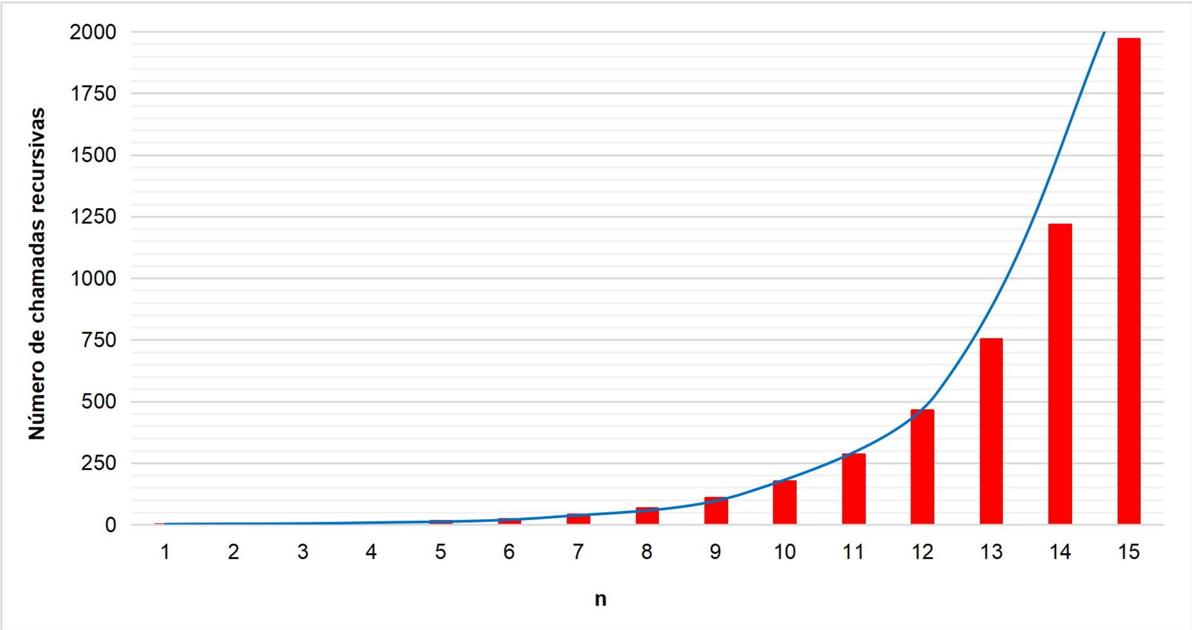

GRÁFICO 1 - n x número de chamadas recursivas (não otimizado)

Para representarmos o GRAF. 1, limitamos os valores de n x número de chamadas recursivas para melhor visualização (n com o valor máximo de 15) . Sendo assim, ao analisar o GRAF. 1, percebemos que o número de chamadas recursivas aumenta exponencialmente conforme o valor que n cresce, seguindo a natureza da implementação recursiva. Cada chamada recursiva envolve recalcular todo os valores para F(n-1) e F(n-2), levando a duplicações de cálculos, o que explica os grandes números gerados das chamadas recursivas.

---

2) É possível otimizar a implementação recursiva, mantendo em um vetor, valores já calculados para a série e reaproveitando esses valores para novos números da série. Altere a implementação realizada no item anterior, considerando essa otimização. Essa forma de otimização é conhecida como programação dinâmica.
    a) Conte o número de chamadas recursivas para cada execução;

Resposta 2 - a)

```
F(1) = 1 (Chamadas recursivas: 1)
F(2) = 1 (Chamadas recursivas: 3)
F(3) = 2 (Chamadas recursivas: 5)
F(4) = 3 (Chamadas recursivas: 7)
F(5) = 5 (Chamadas recursivas: 9)
F(6) = 8 (Chamadas recursivas: 11)
F(7) = 13 (Chamadas recursivas: 13)
F(8) = 21 (Chamadas recursivas: 15)
F(9) = 34 (Chamadas recursivas: 17)
F(10) = 55 (Chamadas recursivas: 19)
F(11) = 89 (Chamadas recursivas: 21)
F(12) = 144 (Chamadas recursivas: 23)
F(13) = 233 (Chamadas recursivas: 25)
F(14) = 377 (Chamadas recursivas: 27)
F(15) = 610 (Chamadas recursivas: 29)
F(16) = 987 (Chamadas recursivas: 31)
F(17) = 1597 (Chamadas recursivas: 33)
F(18) = 2584 (Chamadas recursivas: 35)
F(19) = 4181 (Chamadas recursivas: 37)
F(20) = 6765 (Chamadas recursivas: 39)
F(21) = 10946 (Chamadas recursivas: 41)
F(22) = 17711 (Chamadas recursivas: 43)
F(23) = 28657 (Chamadas recursivas: 45)
F(24) = 46368 (Chamadas recursivas: 47)
```

```
F(25) = 75025 (Chamadas recursivas: 49)
F(26) = 121393 (Chamadas recursivas: 51)
F(27) = 196418 (Chamadas recursivas: 53)
F(28) = 317811 (Chamadas recursivas: 55)
F(29) = 514229 (Chamadas recursivas: 57)
F(30) = 832040 (Chamadas recursivas: 59)
F(31) = 1346269 (Chamadas recursivas: 61)
F(32) = 2178309 (Chamadas recursivas: 63)
F(33) = 3524578 (Chamadas recursivas: 65)
F(34) = 5702887 (Chamadas recursivas: 67)
F(35) = 9227465 (Chamadas recursivas: 69)
F(36) = 14930352 (Chamadas recursivas: 71)
F(37) = 24157817 (Chamadas recursivas: 73)
F(38) = 39088169 (Chamadas recursivas: 75)
F(39) = 63245986 (Chamadas recursivas: 77)
F(40) = 102334155 (Chamadas recursivas: 79)
```

Para o resultado acima, foi considerando que n = 40. A comparação entre os dois conjuntos de resultados mostra claramente a redução do valor das chamadas recursivas e o impacto da otimização na implementação da sequência de Fibonacci, fazendo com que a execução do código melhorasse.

---

b) Gere um gráfico de **`n x número de chamadas recursivas`**. Verifique que é uma função polinomial;

Resposta 2 - b)

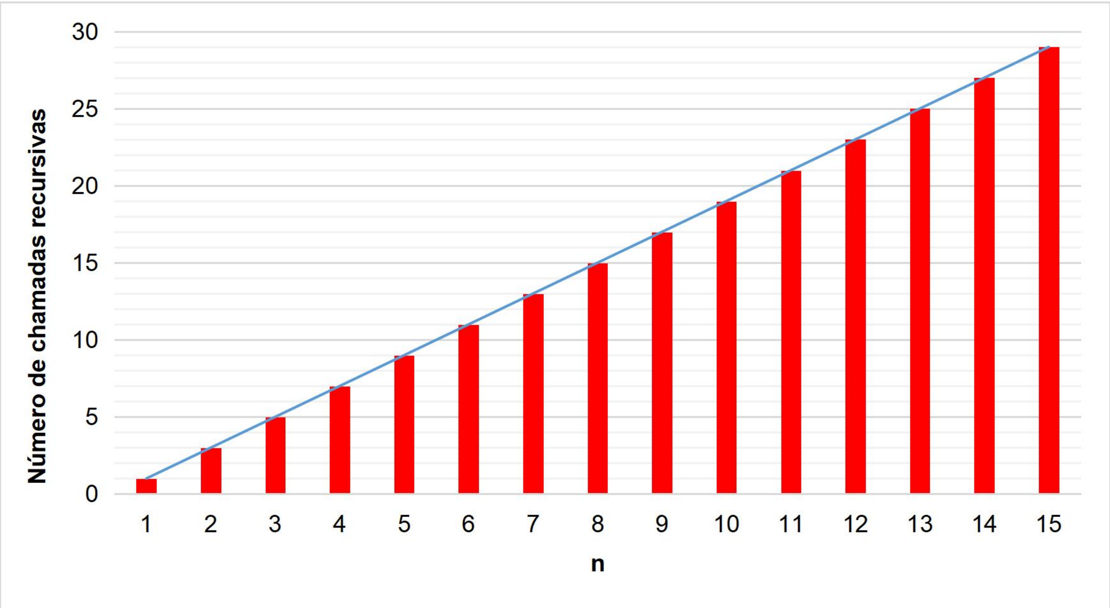

GRÁFICO 2 - n x número de chamadas recursivas (otimizado)

Para representarmos o GRAF. 2, também limitamos os valores de n x número de chamadas recursivas para melhor visualização (n com o valor máximo de 15) . Sendo assim, ao analisar o GRAF. 2, percebemos que o número de chamadas recursivas aumenta conforme o valor que n cresce, seguindo a natureza da implementação recursiva, e a relação existente entre os eixos é uma função polinomial. Cada chamada recursiva não envolve mais recalcular todo os valores para F(n-1) e F(n-2) igual do exercício 1. Nessa implementação otimizada, mantemos em um vetor, os valores já calculados para a série e reaproveitamos esses valores para novos números da série.

Portanto, a comparação entre as implementações recursivas da sequência de Fibonacci evidencia a importância da otimização na eficiência computacional. Enquanto a versão sem otimização apresenta um crescimento exponencial no número de chamadas recursivas, resultando em custos computacionais significativos, a implementação com otimização apresenta um crescimento polinomial, resultando em um bom desempenho computacional.
A otimização do algoritmo evita cálculos redundantes, transformando a abordagem recursiva em uma solução mais eficiente e aplicável a valores maiores de n. Isso destaca a relevância da programação dinâmica como uma ferramenta valiosa para aprimorar o desempenho de algoritmos recursivos em problemas que envolvem subproblemas sobrepostos.